United Electronic Industries

The High-Performance Alternative

UEIPAC-300/600-1G
User Manual

October 2009 Edition
PN Man-UEIPAC-300/600-1G-1009
Version 1.0

© Copyright 1998-2009 United Electronic Industries, Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form by any means, electronic, mechanical, by photocopying, recording, or otherwise without prior written permission.

Information furnished in this manual is believed to be accurate and reliable. However, no responsibility is assumed for its use, or for any infringement of patents or other rights of third parties that may result from its use.

All product names listed are trademarks or trade names of their respective companies.

See the UEI website for complete terms and conditions of sale:

http://www.ueidaq.com/company/terms.aspx

Contacting United Electronic Industries

Mailing Address:

27 Renmar Avenue
Walpole, MA 02081
U.S.A.

For a list of our distributors and partners in the US and around the world, please see http://www.ueidaq.com/partners/

Support:

Telephone:(508) 921-4600
Fax:(508) 668-2350

Also see the FAQs and online “Live Help” feature on our web site.

Internet Support:

Support:support@ueidaq.com
Web-Site:www.ueidaq.com
FTP Site:ftp://ftp.ueidaq.com

Product Disclaimers:

WARNING!

DO NOT USE PRODUCTS SOLD BY UNITED ELECTRONIC INDUSTRIES, INC. AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS.

Products sold by United Electronic Industries, Inc. are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Any attempt to purchase any United Electronic Industries, Inc. product for that purpose is null and void and United Electronics Industries, Inc. accepts no liability whatsoever in contract, tort, or otherwise whether or not resulting from our or our employees' negligence or failure to detect an improper purchase.

Specifications in this document may change without notice. Check with UEI for current status.

Table of Contents

Chapter 1 The UEIPAC-300/600-1G Cube 1

1.1 Organization of Manual 1

1.2 PowerDNA UEIPAC-300/600-1G System 3

1.3 Specifications 5

1.4 Key Features 6

1.5 PowerDNA UEIPAC-300/600-1G Cube 7
1.5.1 Cooling Air Flow 9
1.5.2 DNA Power, CPU, NIC, and I/O Layers 10

1.6 DNA-POWER-1GB Layer 12

1.7 UEIPAC-300/600-1G CPU Layer 15

1.8 DNA-IO-Modules 15

1.9 DC Power Thresholds 15

Chapter 2 Installation and Configuration 16

2.1 Initial Installation Guide 16

2.2 Mounting and Field Connections 19
2.2.1 Physical Dimensions 19

2.3 Wiring 20

Chapter 3 The UEIPAC-300/600-1G Core Module 21

4.1 Device Architecture of the UEIPAC-300/600-1G Core Module 22
3.1.1 Primary Network Interface MII Port – NIC1 23
3.1.2 Secondary Network Interface Port – NIC2 23
3.1.3 RS-232 Port 23
3.1.4 UBS 2.0 Dual Port (Controller and Slave) 23
3.1.5 32MB Flash Memory 23
3.1.6 128MB of SDRAM 23
3.1.7 SYNC Port 23
3.1.8 SD Card 23
3.1.9 LEDs 23
3.1.10 Watchdog Timer With Real-time Clock (Battery Backed) 23

Chapter 4 Real-time Operation with an IOM 24

4.1 Simple I/O 24

4.2 Real-time Data Mapping (RtDmap) 24
4.2.1 Data Replication over the Network 25
4.2.2 RtDmap Functional Description 25
4.2.3 RtDmap Typical Program Structure 27

4.3 Real-time Variable-size Data Mapping (RtVmap) 28
4.3.1 RtVmap Typical Program Structure 33

Appendix A – Field Repacement of Fuses 35

Index 37

List of Figures

Chapter 1 The UEIPAC-300/600-1G Cube 1
1-1 UEI Typical UEIPAC-300/600-1G System 3
1-2 Technical Specifications 5
1-3 Product Features 7
1-4 Typical UEIPAC-300/600-1G Cube with Stack Pulled Out 7
1-5 DNA-IO-Filler Panel for unused slots 8
1-6 UEIPAC-300/600-1G Cube Air Flow 9
1-7 UEIPAC-300/600-1G System Front Panel Arrangement 10
1-8 UEIPAC-300/600-1G Front Panel LEDs 11
1-9 DNA-POWER-1GB DC Power and NIC Layer Front 12
1-10 Functional Block Diagram of DNA-POWER-1GB Module 13
1-11 Functional Block Diagram of UEIPAC-300/600-1G Core Module (2 Boards) 14

Chapter 2 Installation and Configuration 16
2-1 UEIPAC-300/600-1G Front Panel 16
2-2 Physical Dimensions of UEIPAC-300/600-1G Cubes 19
2-3 System Configuration with LAN Switch 20

Chapter 3 The UEIPAC-300/600-1G Core Module 21
3-1 UEIPAC Core Module – DNA-CPU/NIC-1G 21
3-2 PowerDNA Core Module – DNA-CPU/NIC-1G 21
3-3 FreeScale PowerPC CPU/NIC Controller Architecture 22

Chapter 4 Real-time Operation with an IOM 24
4-1 DMap Operation 24

Appendix A – Field Repacement of Fuses 35
A-1 Location of Fuse for PL-61x, PL-62x, and PL-63x Boards 1
A-2 Location of Fuses for DNR-POWER-1GB Board 2

Index 37

Chapter 1 The UEIPAC-300/600-1G Cube

This document describes the features, performance specifications, and operating functions of the UEIPAC-300/600-1G Gigabit Ethernet Cube data acquisition system. The system is designed for use with an Ethernet Gigabit 1000 Base-T communication network.

1.1 Organization of Manual

This UEIPAC-300/600-1G User Manual is organized as follows:

- **DNA- UEIPAC-300/600-1G Gigabit Ethernet Cube System**
  This chapter provides an overview of a UEIPAC system, component modules, features, accessories, and a list of all items you need for initial operation.
- **Installation and Configuration.**
  This chapter summarizes the recommended procedures for installing, configuring, starting up, and troubleshooting a UEIPAC-300/600-1G system.
- **The UEIPAC-300/600-1G Core Module (CPU/NIC)**
  This chapter describes the UEIPAC-300/600-1G CPU/NIC module, which contains a PowerPPC CPU and a GigE Network Interface Module.
- **Real-time Operation with an IOM**
  This chapter discusses operation of the UEIPAC-300/600-1G system under control of a real-time operating system.
- **Appendix A – Field Replacement of Fuses**
- **Index**
  This is an alphabetical listing of topics covered in the manual, identified by page number.

© Copyright 2009
United Electronic Industries, Inc.

Manual Conventions

To help you get the most out of this manual and our products, please note that we use the following conventions:

Tips are designed to highlight quick ways to get the job done, or reveal good ideas you might not discover on your own.

NOTE: Notes alert you to important information.

CAUTION! *advises you of precautions to take to avoid injury, data loss, and damage to your boards or a system crash.*

Text formatted in **bold** typeface generally represents text you should enter verbatim. For instance, it can represent a command, as in the following example: “You can instruct users how to run setup using a command such as **setup.exe**.”

Before plugging any I/O connector into the Cube or Board, be sure to remove power from all field wiring. Failure to do so may cause severe damage to the equipment.

Usage of Terms

In this document, the terms “module” and “layer” are used interchangeably. In UEIPAC Cubes, a “layer” refers to a data acquisition module (circuit board), which is typically assembled with other modules into a “multi-layer” stack for insertion into a UEIPAC Cube housing.

1.2 PowerDNA UEIPAC-300/600-1G System

The PowerDNA UEIPAC-300/600-1G product is a GigE version of the popular PowerDNA Cube Ethernet-based Data Acquisition System. The UEIPAC-300-1G system houses a PowerDNA Gigabit Ethernet data acquisition system in a 5-slot cube housing that can accept up to 3 I/O layers accessible from the front. Up to 4 cube systems may be mounted in a rack DNA-19RACKW accessory assembly.

The UEIPAC-600-1G houses a PowerDNA Gigabit Ethernet data acquisition system in a 8-slot cube housing that can accept up to 6 I/O layers accessible from the front. Both 5- and 8-slot GigE Cubes are also available in fiber optic versions: (FPPC5-1G and FPPC8-1G).

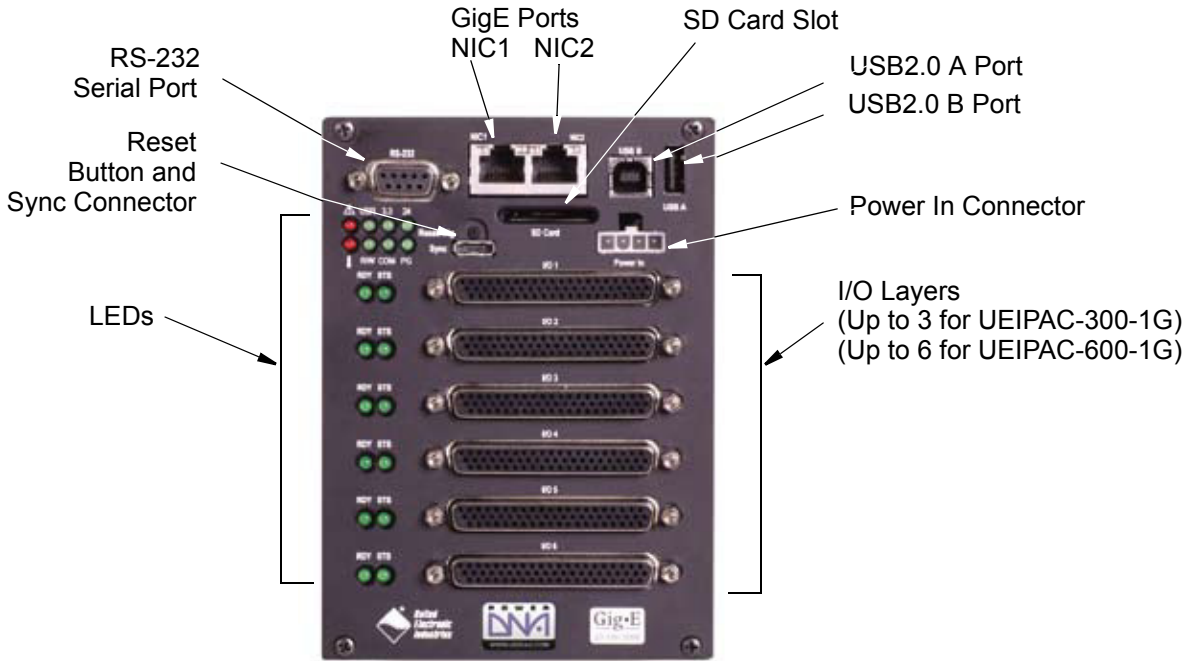

Figure 1-1. UEI Typical PowerDNA UEIPAC-300/600-1G System

As illustrated in **Figure 1-4** and **Figure 1-7**, a standard UEIPAC-300-1G system consists of the following modules:

- One 5-slot or 8-slot Cube Housing
- One DNA-POWER-1GB DC Power Layer (topmost slot)
- One DNA-PPC-1GB CPU layer (second slot)
- Up to 3 (UEIPAC-300-1G) or 6 (UEIPAC-600-1G) DNA-IO-FILLER panels (one for each unused I/O slot)
- DNR-PSU-24-100 100-Watt, 120/230 VAC to +24VDC External Power Supply with cable and Molex connector for plug-in to the Molex Power In connector on the front panel.

To configure a complete data acquisition system, insert up to 3 (or 6) DNA I/O layers into the Cube housing, which may be specified in any combination of the following types:

- DNA-AI-201, -202, 205, 207, -208, -211, -224, -225

© Copyright 2009
United Electronic Industries, Inc.

- DNA-AO-308, -308-350, -308-353, -308-420, -332, -333
- DNA-DIO-401, -402, -403, 404, -405, 406, -416, -432, -433, -448
- DNA-CT-601, DNA-QUAD-604
- DNA-SL-501, DNA-CAN-503
- DNA-429-566, DNA-429-512
- DNA-GPS
- Any future additions to the PowerDNA I/O module product line

Note: Refer to www.ueidaq.com for a description of each I/O module.

© Copyright 2009
United Electronic Industries, Inc.

1.3 Specifications

Figure 1-2 lists the technical specifications of a PowerDNA UEIPAC-300/600-1G Cube.

Technical Specifications:

Standard Interfaces	
Gigabit Ethernet	Two independent 1000/100/10Base-T interfaces, each with a unique IP address (connected via standard RJ-45 connectors)
USB 2.0	Two ports, one controller, one slave
Config/General	RS-232, 9-pin "D"
Sync	Custom cable to sync multiple cubes
I/O Slots Available	
DNA-PPC8g	6 slots
DNA-PPC5g	3 slots
Host Communications	
Distance from host	100 meters max, CAT5+ cable
Ethernet data transfer rate	20 megabyte per second
Analog data transfer rate	>6 megasample per second. Capable of sustained transfer of any cube configuration
DMAP I/O mode	update 1000 I/O channels (analog and/or digital) in less than 1 millisecond, guaranteed
Processor	
CPU	Freescale 8347 series, 400 MHz, 32-bit
Memory	128 MB (not including on-board Flash)
Status LEDs	Attention, Read/Write, Power, Communications Active
Environmental	
Temp (operating)	Tested to -40 °C to 85 °C
Temp (storage)	-40 °C to 100 °C
Humidity	0 to 95%, non-condensing
Vibration	
(IEC 60068-2-64)	10-500 Hz, 5 g (rms), Broad-band random
(IEC 60068-2-6)	10-500 Hz, 5 g, Sinusoidal
Shock	
(IEC 60068-2-27)	50 g, 3 ms half sine, 18 shocks at 6 orientations; 30 g, 11 ms half sine, 18 shocks at 6 orientations
Altitude	70,000 feet, maximum
MTBF	300,000 hours
Physical Dimensions	
DNA-PPC5g	4.1" x 4.0" x 4.0"
DNA-PPC8g	4.1" x 4.0" x 5.8"
Power Requirements	
Voltage	9 - 36 VDC (AC adaptor included)
Power Dissipation	13 W at 24 VDC (not including I/O boards)

Figure 1-2. Technical Specifications

© Copyright 2009
United Electronic Industries, Inc.

1.4 Key Features

The following table is a list of key features of a UEIPAC-300/600-1G PowerDNA Cube.

Easy to configure and deploy

- Over 30 different I/O boards available
- Built-in signal conditioning
- Gigabit Ethernet based (100/10Base-T compatible)
- Flange kit for mounting to wall/flat surface
- DIN rail and Rack Mount kits
- Attach style carrying case available for portable deployments
- Standard "Off-the-shelf" products and delivery

True Real-time Performance

- 1 msec updates guaranteed with 1000 I/O
- Up to 6 million samples per second
- Use QNX, RTX, RT Linux, RTAI Linux, LabVIEW RT

Flexible Connectivity

- Dual 1000Base-T Gigabit Ethernet ports with independent IPs
- Dual USB 2.0 controller ports
- 10/100Base-FX Fiber interface available (see DNA-FPPC family)
- Supports WIFI / GSM / Cell networks

Compact Size:

- 4.1" x 4" x 5.8 Cube holds 6 I/O boards
- 4.1" x 4" x 4" Cube holds 3 I/O boards
- 150 analog inputs per cube,
- 192 analog outputs per cube
- 288 digital I/O bits per cube.
- 48 counter/quadrature channels per cube
- 72 ARINC 429 ports per cube
- 24 Serial or CAN ports per cube

Low Power:

- Less than 13 watts per cube (not including I/O boards0
- AC, 9-36 VDC or battery powered.

Stand alone and Data Logger Modes

- DNA-PPC-G series Cubes can be upgraded with UEI-LOGGER series capabilities
- DNA-PPC-G series Cubes can be upgraded to the Linux based UEIPAC Programmable Automation Controller

Rugged and Industrial:

- All Aluminum construction
- Operation tested from -40°C to 85°C
- Vibration tested to 5 g, (operating)
- Shock tested to 50 g (operating)
- All I/O isolated from Cube and host PC.
- Operation to 70,000 feet

Outstanding Software support

- Windows, Linux, RT Linux, Windows RT, RTX, VXworks and QNX operating systems
- VB, VB .NET, C, C#, C++, J#

Figure 1-3. Product Features

© Copyright 2009
United Electronic Industries, Inc.

1.5 PowerDNA UEIPAC-300/600-1G Cube

Each UEIPAC-300/600-1G cube contains a two-layer Core Module with status indicating LEDs, serial port, A and B USB ports, two GigE network interface ports, an SD card reader, power circuits, reset button/sync interface. The Core Module contains a GigE CPU and two Network Interface Control (NIC) ports, one for controlling up to 6 I/O layers mounted in the cube, and another for miscellaneous use. The device-specific I/O boards are functionally identical to the corresponding modules for the PowerDNR RACKtangle.The only differences between the two types relate to the mounting arrangements.

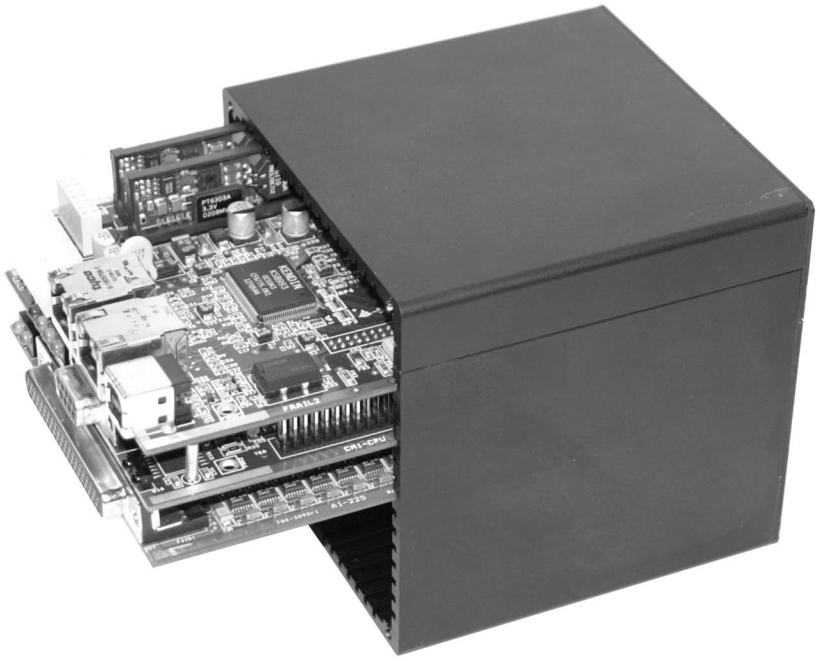

Figure 1-4. Typical UEIPAC-300/600-1G Cube with Stack Pulled Out

As shown in **Figure 1-4** and **Figure 1-7**, the UEIPAC-300/600-1G enclosure is designed to house the following items:

- One isolated DNA-POWER-1GB DC/DC Power Module/Power Monitor with RJ-45 connectors for NIC2 and NIC2, DB-9 connector for a serial port, and A and B USB controller/slave ports.
- One DNA-PPC-1GB CPU module with 8 indicating LEDs, sync connector, reset pushbutton, SD card slot, and a 4-pin Molex Power In connector.
- Up to 3 (UEIPAC-300-1G) or 6 (UEIPAC-600-1G) PowerDNA I/O layers (boards) functionally identical to PowerDNR I/O boards but designed for mounting in a DNA cube housing
- Blank filler panels for all unused slots
- One (for UEIPAC-300-1G) or two (for UEIPAC-600-1G) 8-volt cooling fans mounted on the rear cover of the Cube

The cube itself is a rigid, extruded aluminum, mechanical structure with complete EMI shielding. Unused slots are filled with blank filler panels. The DC power module provides output voltages of 24, 3.3, 2.5, 1.5, and 1.2 VDC for the logic/CPU and 8 VDC to power the cooling fans.

© Copyright 2009
United Electronic Industries, Inc.

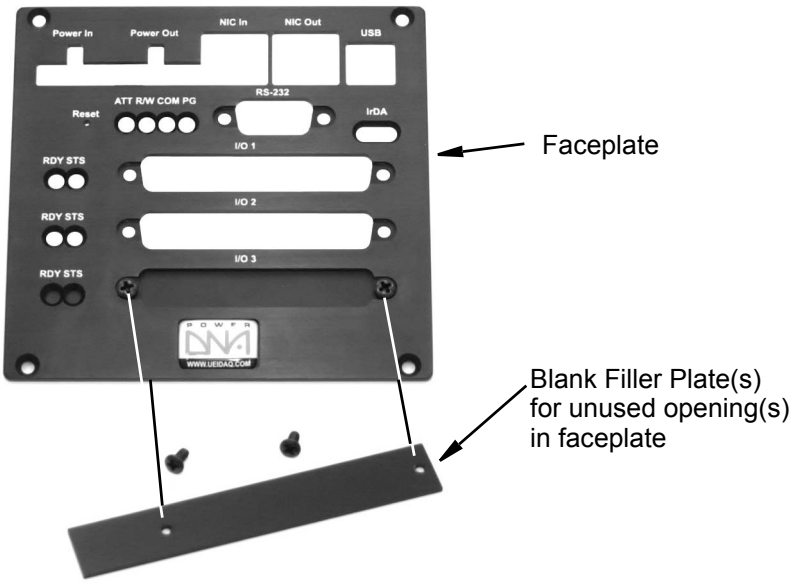

Figure 1-5. DNA-IO-Filler Panel for unused slots

1.5.1 Cooling Air Flow

As shown in **Figure 1-6**, cooling is drawn into the rear of the enclosure, routed forward over the electronic circuit boards, up to the top of the enclosure, and then out the top rear of the enclosure. The system is designed to maintain positive pressure cooling within the enclosure at all times.

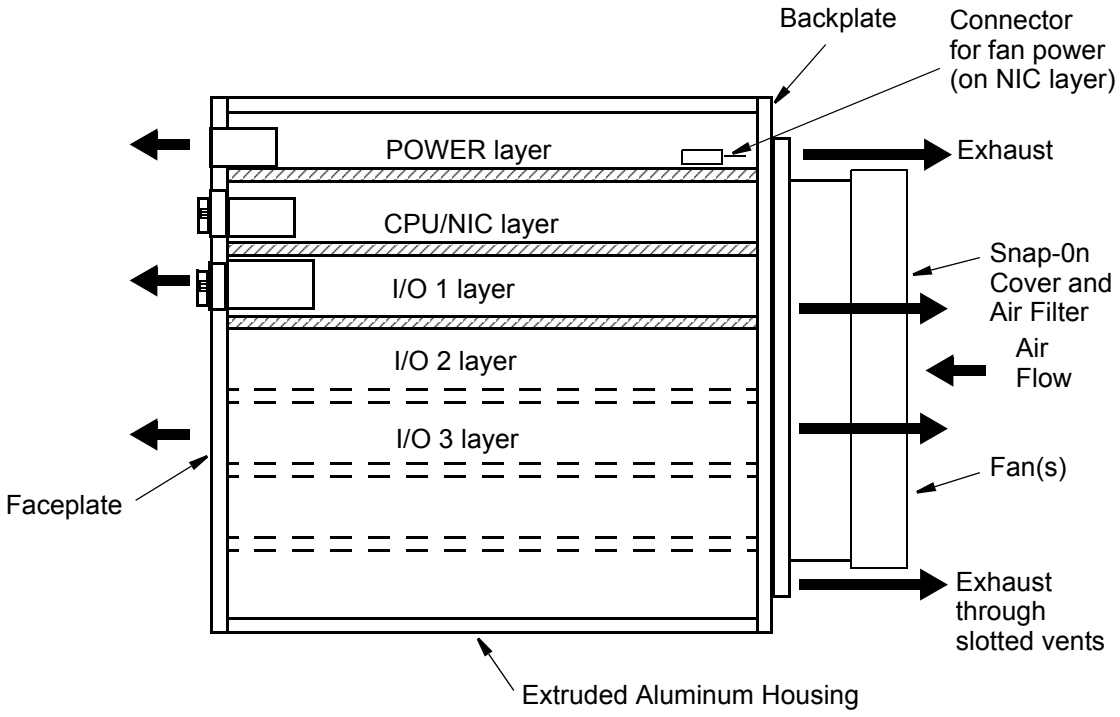

Figure 1-6. UEIPAC-300/600-1G Cube Air Flow

1.5.2 DNA Power, CPU, NIC, and I/O Layers

This section describes the basic modules included in every UEIPAC-300/600-1G system – the CPU/NIC layer, the DC power layer, and I/O layers.

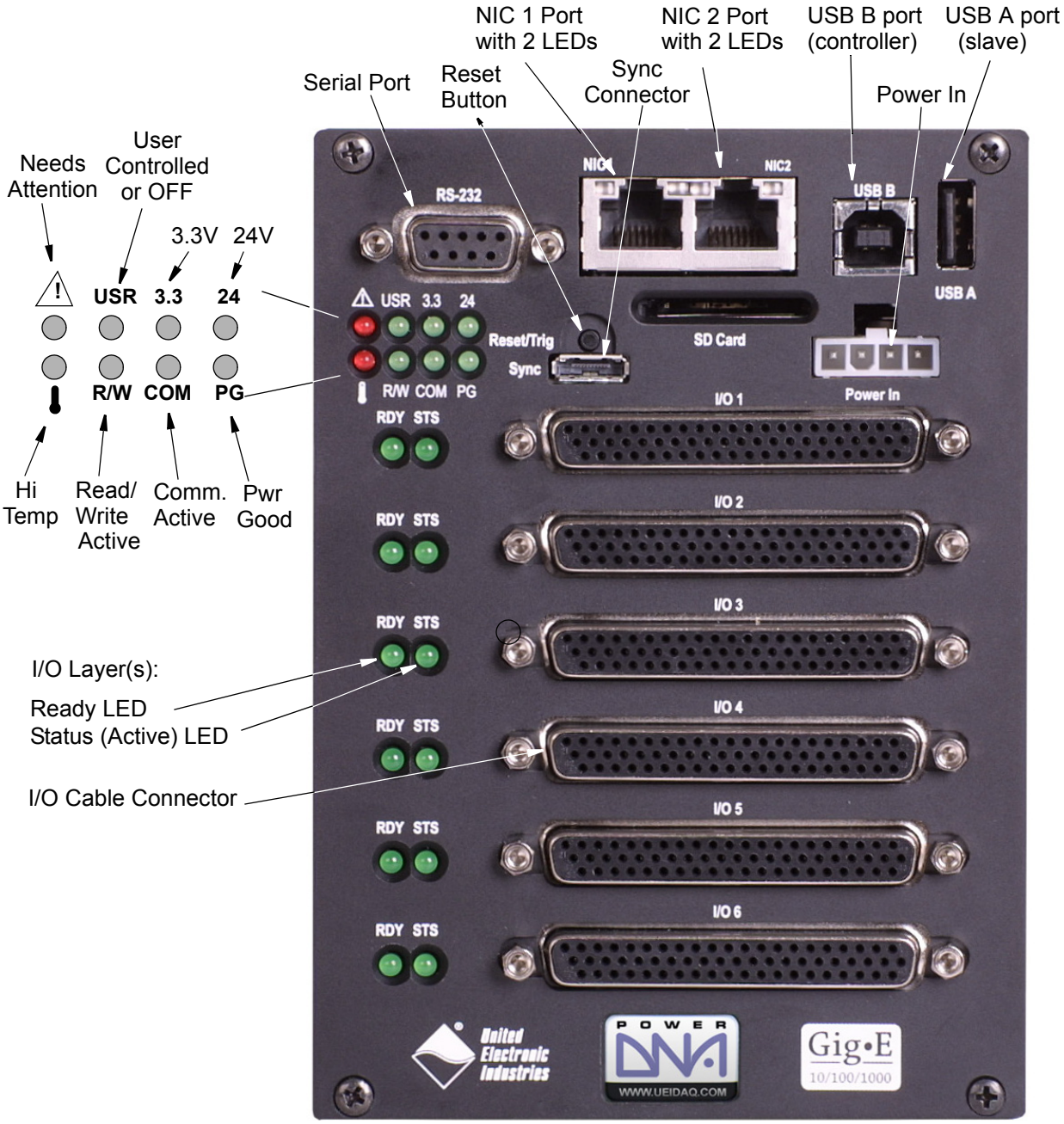

Figure 1-7. UEIPAC-300/600-1G System Front Panel Arrangement

Figure 1-8 describes the conditions indicated by the LEDs on the front of each module in the rack.

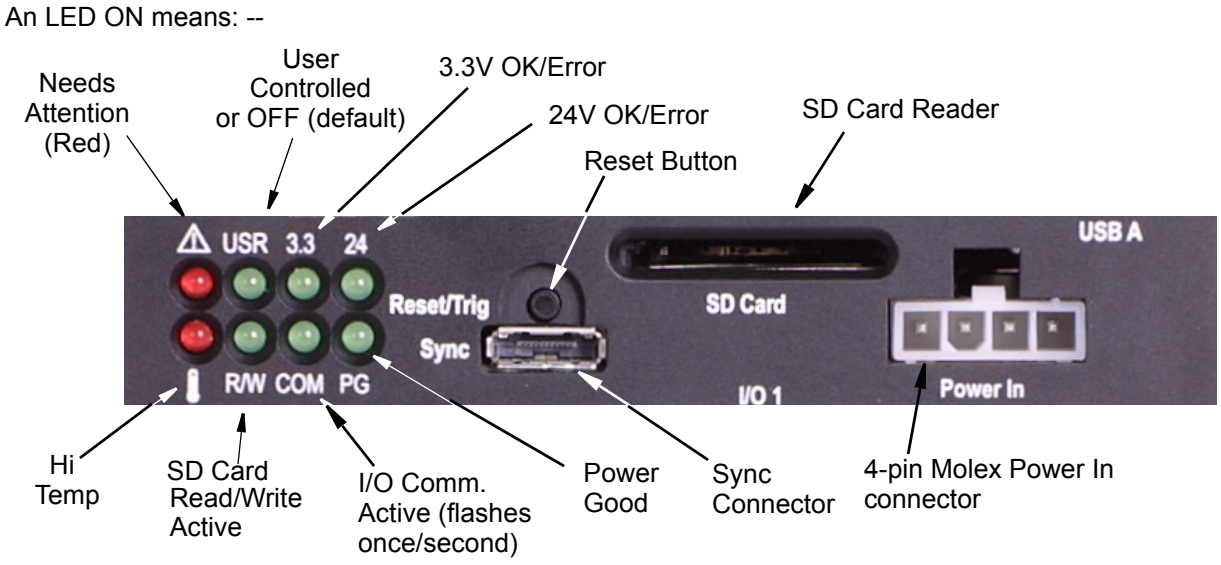

Figure 1-8. UEIPAC-300/600-1G Front Panel LEDs

Figure 1-8 describes the meanings of various states of the indicating LEDs mounted on the front panel of the Cube. The LEDs are physically mounted on the CPU Layer, which in located in the second topmost layer position form the top of the Cube.

A temperature sensor mounted on the DNA-Power-DC Layer monitors internal temperature continuously, turning fan(s) on if the internal temperature exceeds 45°C, off if it falls below 45°C, and shutting down power if a high limit is exceeded.

© Copyright 2009
United Electronic Industries, Inc.

1.6 DNA-POWER-1GB Layer

The DNA-POWER-1GB Layer has a dedicated DC/DC source available for use with the UEIPAC-300/600-1G Cube. It is always mounted in the topmost slot of the cube and is recognized on the PowerDNA bus with an ID of 0x020 at address 0xA00C0000.

The non-isolated side (NIS) logic complies with full common logic interface (CLI) implementation. The key features of the unit are:

- Input power — 9-36 VDC 80W maximum, protected by resettable fuses and EMI chokes
- Output power sources (all with greater than 90% efficiency)

  24V, 1A (24W)
  3.3V, 5A (16.5W, including the 2.5V derived voltage)
  2.5V, 3A (derived from 3.3V source)
  1.5V, 5A, (7.5W, including the 1.2V derived voltage)
  8V, 0.5A (4W for fans)
- DC/DC for 24V, 3.3V, and 1.5V are synchronized from the single spread-spectrum clock source in the CPU/NIC Core Module for low EMI noise level
- Output Power provided for Fan control (Forced ON) and status ON/OFF
- Monitoring and LED indicators (1% accuracy, 0.25Hz update rate, mounted on CPU Layer)
  for:

  – All output voltages
  – Input current for the 9-36VDC for the DNA Cube Housing
  – All voltages from the NIC/Power Module (24V, 3.3V, 2.5V)
  – Temperature of the DNA-PPCx Cube Housing and layers
- Onboard FPGA logic chip is CYCLONE EP1C3/C6T144
- TI MSP4300 microcontroller used for logic reprogramming
- Provides 9-36VDC for all modules from an external power source

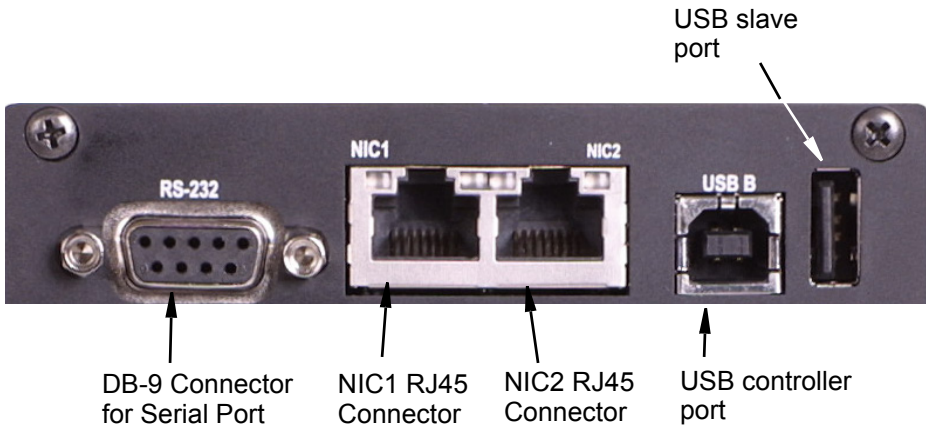

Figure 1-9. DNA-POWER-1GB DC Power and NIC Layer Front

A functional block diagram of the DNA-POWER-1GB Module is shown in **Figure 1-10** below.

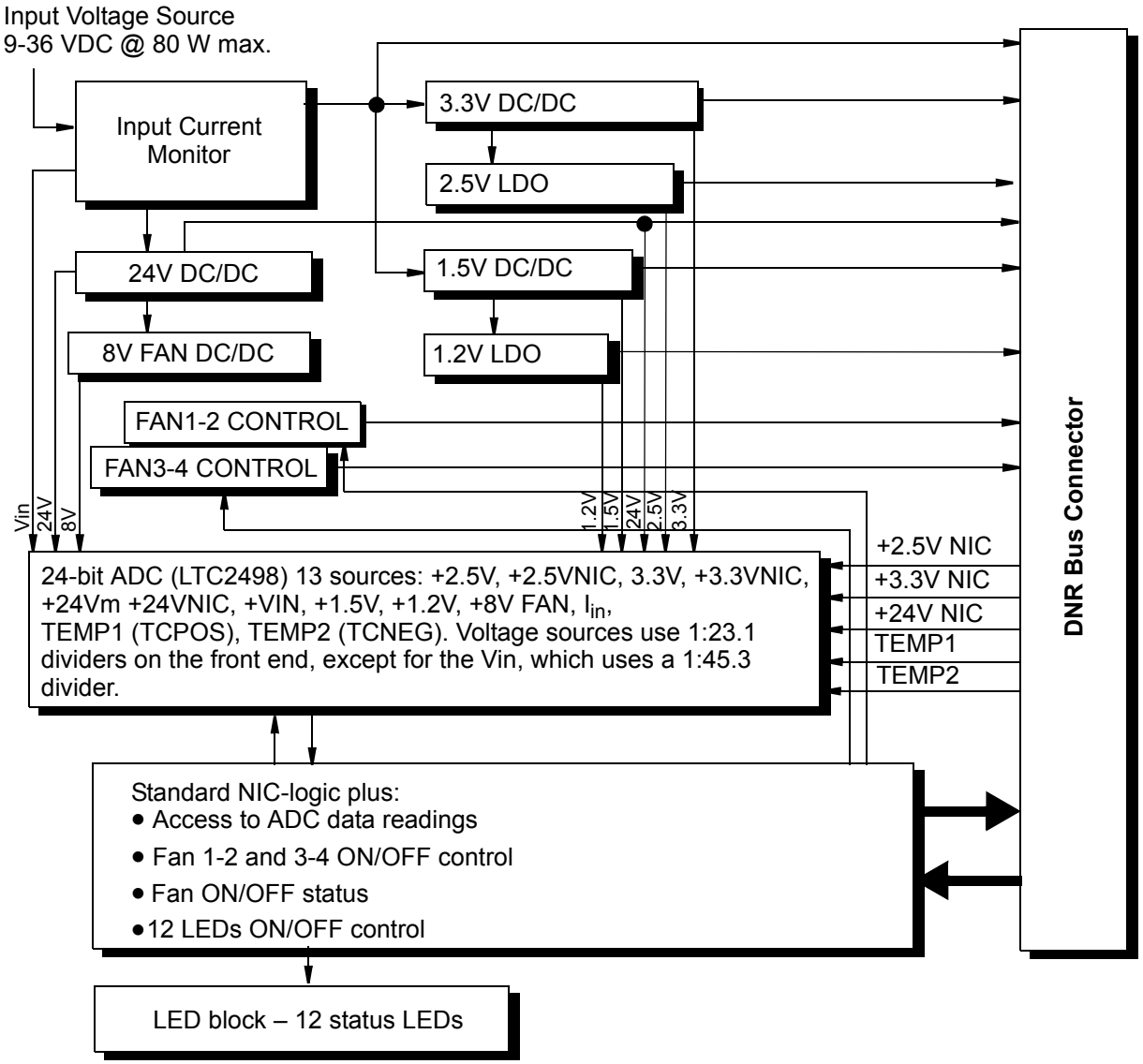

Figure 1-10. Functional Block Diagram of DNA-POWER-1GB Module

As shown in **Figure 1-10**, the DNA-POWER-1GB Module operates as follows:

A 9-36VDC voltage input (Vin) from an external source is connected to the board through a resettable fuse. The board monitors the input current and passes Vin to the DNA bus as Vout. Vout also is connected to DC/DC converters that produce 24 VDC, 3.3VDC and 1.5VDC output voltages, which are also placed on the DNA bus. Both 3.3 and 1.5VDC voltages are connected to low dropout regulators that, in turn, generate the 2.5VDC and 1.2VDC output voltages on the bus. The 24VDC source is fed to a low dropout regulator that produces 8VDC to drive the cooling fans (through fan controller chips).

The input current and all output voltages, including the +2.5, +3.3, and +24VDC from the NIC module, plus signals from the temperature sensor mounted within the enclosure, are input to a 24-bit delta-sigma A/D converter. Except for Vin, the voltage sources use 1:23.1 dividers on the front end. Vin uses a 1:45.3 divider.

Figure 1-11 shows a functional block diagram of a UEIPAC-300/600-1G Core Module, which consists of a CPU/NIC Layer and a DNA-POWER-1G DC layer, assembled in a UEIPAC-300/600-1G Cube.

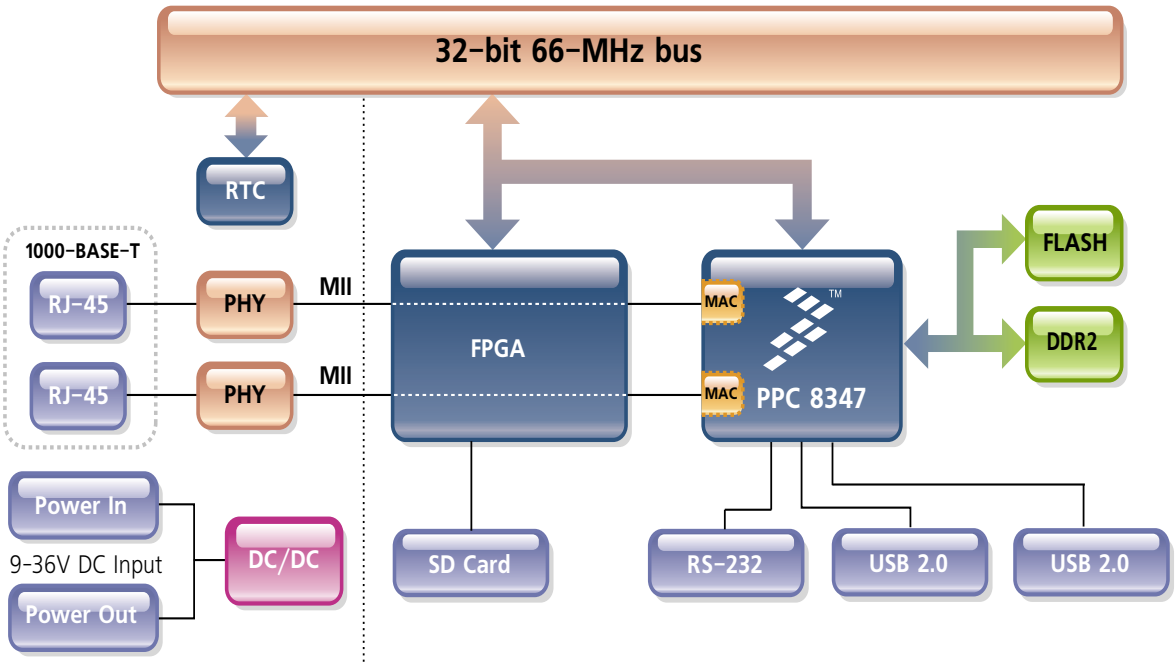

Figure 1-11. Functional Block Diagram of UEIPAC-300/600-1G Core Module (2 Boards)

As shown in **Figure 1-11**, the Core Module uses a Freescale (formerly Motorola) PowerPC 8347 400 MHz, 32-bit processor, with Flash Memory and 128MB DDR2 Memory. Two independent gigabit Ethernet Network Interface Controllers (NICs) are provided, each with its own IP address. One, usually NIC1, is configured as a main control port and the other, NIC2, as a general purpose port. Either port may be assigned either function.An FPGA provides the logic for controlling all system operation and offers a convenient mechanism for modifying and extending system functions.

The CPU Layer (second topmost layer position) also provides 8 indicating LEDs, a recessed manual reset button, sync bus connector for connection to other Cubes and systems, SD Card Read/Write unit, and 4-pin Molex connector for 9-36 VDC incoming power.

The associated DNA-POWER-1G module (topmost layer position) has an RS-232 serial port with DB-9 front panel connector, two RJ-45 connectors (each with two green LEDs) for the NIC1 and NIC2 Ethernet ports, two USB 2.0 ports (A and B connectors) for USB host and slave connections, plus a board-mounted connector at the rear of the board for supplying power to cooling fans on the rear cover of the Cube.

A temperature sensor mounted on the POWER layer monitors temperature within the Cube above the CPU. The system turns on the fan(s) if temperature exceeds 45° C and shuts down power to the Cube if a high limit is exceeded.

1.7 UEIPAC-300/600-1G CPU Layer

The UEIPAC-300/600-1G CPU/NIC Layer contains a PowerPC 8347 CPU and associated Network Interface Control (NIC) logic that controls all Ethernet communication functions. The unit has a dual 1GB Ethernet module.

1.8 DNA-IO-Modules

UEI I/O modules are available either as PowerDNA versions for use with UEIPAC-300/600-1G Cubes or as PowerDNR versions for installation in a DNR rack enclosure. Both versions are functionally identical. The only difference between them is the physical mounting arrangement.

For detailed electrical specifications and user instructions for a specific DNA I/O board, refer to the datasheets and User Manuals for that specific layer. These documents are available for examination and download from the UEI website at www.ueidaq.com.

1.9 DC Power Thresholds

Table 5-1 lists the DC power threshold specifications for UEIPAC-300/600-1G Cubes.

Table 5-1 DC Power Thresholds for UEIPAC-300/600-1G Cubes

	Backplane Power Rail Voltages	Turn-on Voltage, V[1]	Reset Voltage, V	Turn-off Voltage, V[2]	Notes
Logic power supply	+3.3V, +2.5V, +1.5V, +1.2V	7.5	7.2 (When Vin is below 7.2V, a voltage reset puts all layers into reset mode.)	7.0	Supplies power to all CPUs and FPGAs. Cube can communicate with Ethernet when CPU is functional
Analog power supply	+24V	8.5	-	7.8	Analog power supply is used as a regulated source for on-layer DC/DCs on most layers
Fan power supply	+12V	8.5	-	8.4	
On-layer DC/DCs that use input power	+VIn	7.8-8.8	-	7.5-8.5	Varies with layer type.

1. Turn-on, V: The value of Vin at which the corresponding DC/DCs are turned on.
2. Turn-off, V: The value of Vin at which the corresponding DC/DCs are turned off.

NOTE: A DNA-PPC-1GB core module consumes only 70mW when Vin is below 7V.

Chapter 2 Installation and Configuration

Installation consists of:

- UEIPAC-300/600-1G hardware setup
- Software package installation
- Configuration

2.1 Initial Installation Guide

This section describes the procedure recommended for performing an initial hardware and software setup when you first receive a UEIPAC-300/600-1G system.

STEP 1: Unpack and verify package contents.

- UEIPAC-300/600-1G Cube
- 24V DC Power Supply
- Ethernet Cable
- Serial (RS-232) Cable
- UEIPAC SDK CD-ROM

Please do not “power up” or connect your Ethernet cable to the cube until instructed to do so. The front panel of the UEIPAC Cube is shown in **Figure 2-1**.

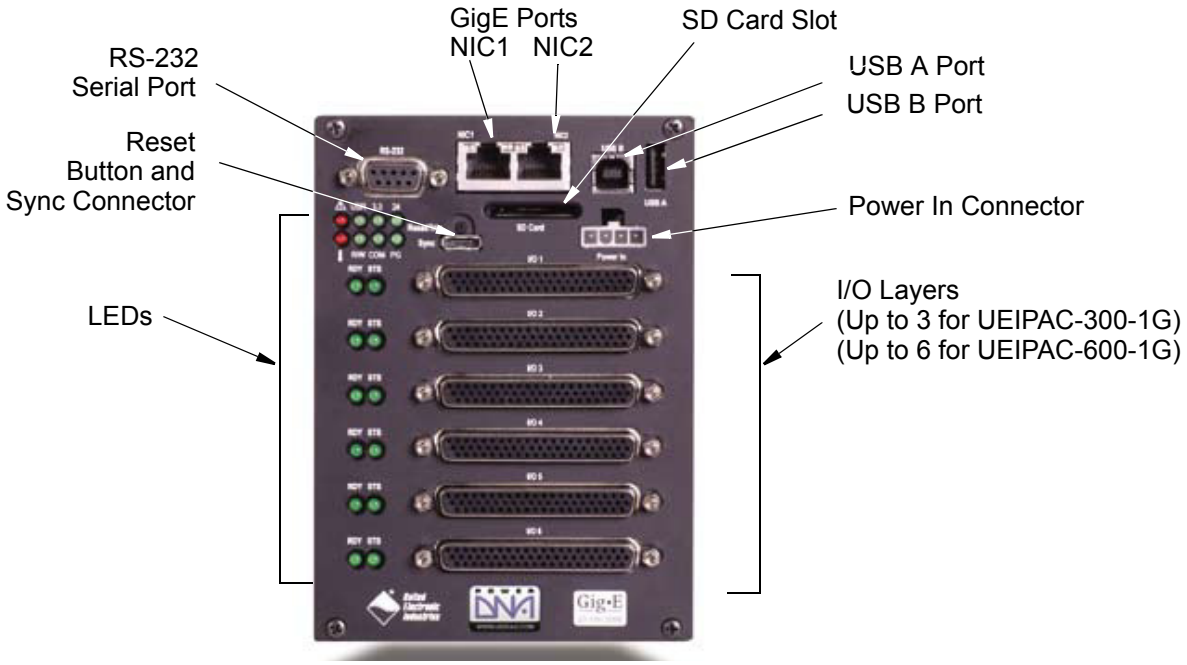

Figure 2-1. UEIPAC-300/600-1G Front Panel

STEP 2: Connect the serial port.

Connect the serial cable to the serial port on the cube and the serial port on your PC.

You will need a serial communication program:

- Windows: ucon, MTTTY or HyperTerminal.
- Linux: minicom or cu (part of the uucp package).

The UEIPAC I/O module uses the serial port settings: 57600 bits/s, 8 data bits, 1 stop bit, and no parity.

Run your serial terminal program and configure the serial communication settings accordingly.

STEP 3: Power-up the UEIPAC.

Connect the DC output of the power supply (24VDC) to the “Power In” connector on the UEIPAC cube and connect the AC input on the power supply to an AC power source.

You should see the booting process on your screen:

```
U-Boot 1.1.4 (Jan 10 2006 - 19:20:03)

CPU:   MPC5200 v1.2 at 396 MHz
       Bus 132 MHz, IPB 66 MHz, PCI 33 MHz

Board: UEI PowerDNA MPC5200 Layer
I2C:   85 kHz, ready
DRAM:  128 MB
Reserving 349k for U-Boot at: 07fa8000
FLASH:  4 MB
In:    serial
Out:   serial
Err:   serial
Net:   FEC ETHERNET

…

BusyBox v1.2.2 (2006.11.03-19:16+0000) Built-in
shell (ash)
Enter 'help' for a list of built-in commands.

~ #
```

You can now navigate the file system and enter standard Linux commands such as ls, ps, cd…

STEP 4: Configuring the IP Address.

Your UEIPAC cube is configured at the factory with the IP address 192.168.100.2 to be part of a private network.

You can change the IP address for the current session using the command:

```
setip <new IP address>
```

To make the change permanent, edit the file /etc/rc.sh and change the line that calls `setip`.

STEP 5: Connecting through Telnet.

Once the IP address is configured, you shouldn't need the serial port anymore. You can use telnet to access the exact same command line interface.

Type the following command on your host PC, then login as "root". The password is "root".

```
telnet <Cube's IP address>
```

Type the command "exit" to logout.

STEP 6: Setting up your Development System

A development system is composed of the software tools necessary to create an embedded application targeting Linux on a PowerPC processor.

The development tools can run on a Linux PC or on a Windows PC using the Cygwin environment.

It contains the following:

- GCC cross-compiler targeting the PowerDNA IO module PPC processor.
- GNU toolchain tools such as make.
- Standard Linux libraries such as glibc.
- PowerDNA library to access the various PowerDNA data acquisition devices

Windows Host

If you don't have Cygwin already installed, download and run the installer "setup.exe" from http://www.cygwin.com.

Running setup.exe will install or update Cygwin. We need the packages from the following categories:

- Archive: tools to create and read archives files such as zip, bx2 and tar.
- Admin: administration tools and disk utilities.
- Devel: Development tools such as make and gcc.
- Net: Network utilities such as ftp, tftp and telnet.

Insert the "UEIPAC SDK" CDROM in your CD drive. Then open a cygwin command line shell.

Go to the CD's root directory (the example below assumes that the CD-ROM is the D: drive) and run the installation script:

```
cd /cygdrive/d
./install.sh
```

Linux Host

Insert the “UEIPAC SDK” CDROM in your CD drive. You might need to mount it if your Linux distribution doesn’t detect the CDROM automatically.

To mount it, type:

```
mount /dev/cdrom /mnt/cdrom
cd /mnt/cdrom
```

Run the installation script

```
./install.sh
```

2.2 Mounting and Field Connections

You can mount the Cube on a flat horizontal surface such as a tabletop or floor, a flat vertical surface such as a wall, or in a standard 19-inch rack. For horizontal surface mounting, specify a flange accessory and secure the case directly to the surface. For mounting on a vertical wall surface, specify a 19RACKW accessory with DIN rail and attach the assembly to a standard 19-inch rack.with screws.

2.2.1 Physical Dimensions

The housing used in a UEIPAC-300/600-1G cube consists of an extruded aluminum box with slotted guides plus a faceplate and rear cover. A UEIPAC-300-1G can accept 3 I/O layers and a UEIPAC-600-1G can accept up to 6 I/O layers.The physical dimensions of the two enclosures are shown below in **Figure 2-2**.

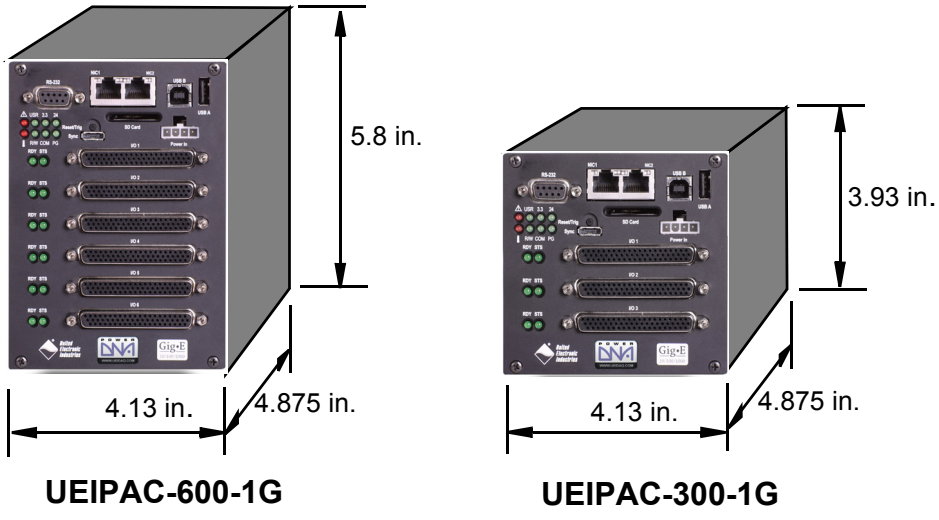

Figure 2-2. Physical Dimensions of UEIPAC-300/600-1G Cubes

2.3 Wiring

1000Base-T Wiring Configurations

A typical wiring configuration for a 1000Base-T network is shown in the following figure.

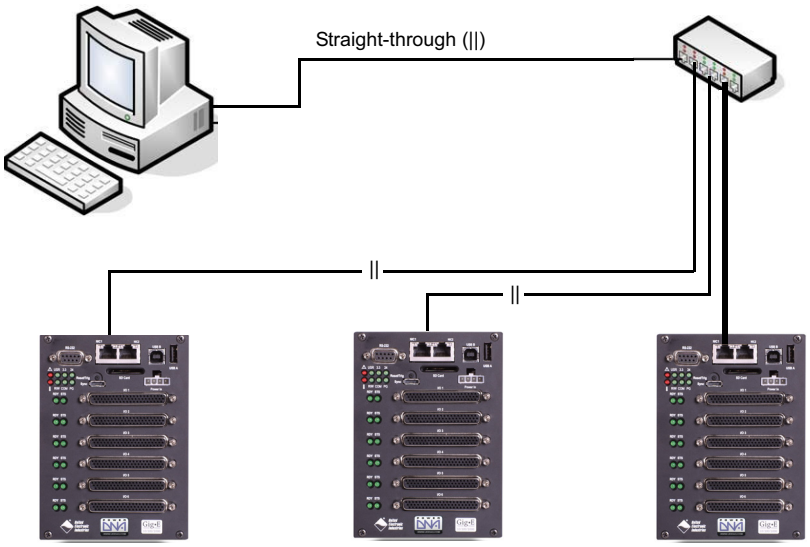

Figure 2-3. System Configuration with LAN Switch

Chapter 3 The UEIPAC-300/600-1G Core Module

This chapter focuses on the device architecture of the Core Module, not I/O modules.

The top two slots of a UEIPAC-300/600-1G Cube housing are occupied by the PowerDNA Core Module, called the DNA-CPU-1G and DNA-NIC-1G boards.

The Core Module consists of a Freescale (formerly Motorola) MPC8347 32-bit 400 MHz CPU and peripheral devices (USB 2.0, RS-232, NIC, SD, etc) for use with a Gigabit Ethernet communication network and an internal 66 MHz 32-bit common logic interface bus. The NICs are copper (1000BaseT) interfaces. The core module has an RS-232 port used for configuration and also two USB 2.0 ports (controller and slave) for general purpose us. LEDs on the front panel of each module indicate the current operating status of the device.

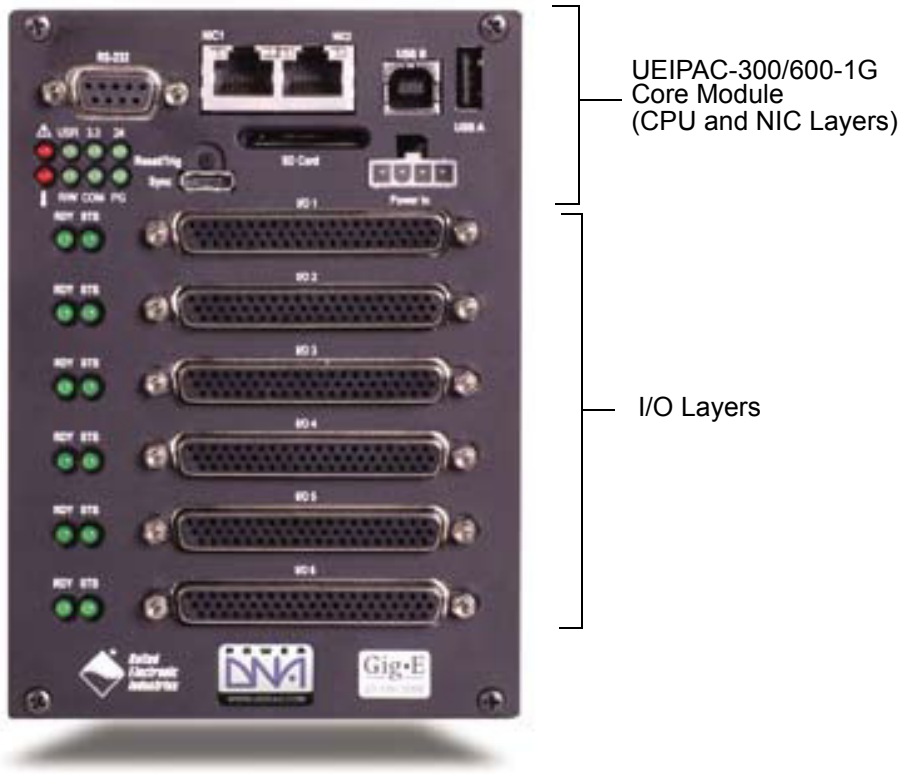

Figure 3-1. UEIPAC Core Module – DNA-CPU/NIC-1G

4.1 Device Architecture of the UEIPAC-300/600-1G Core Module

The UEIPAC-300/600-1G Core Module architecture can be represented as follows:

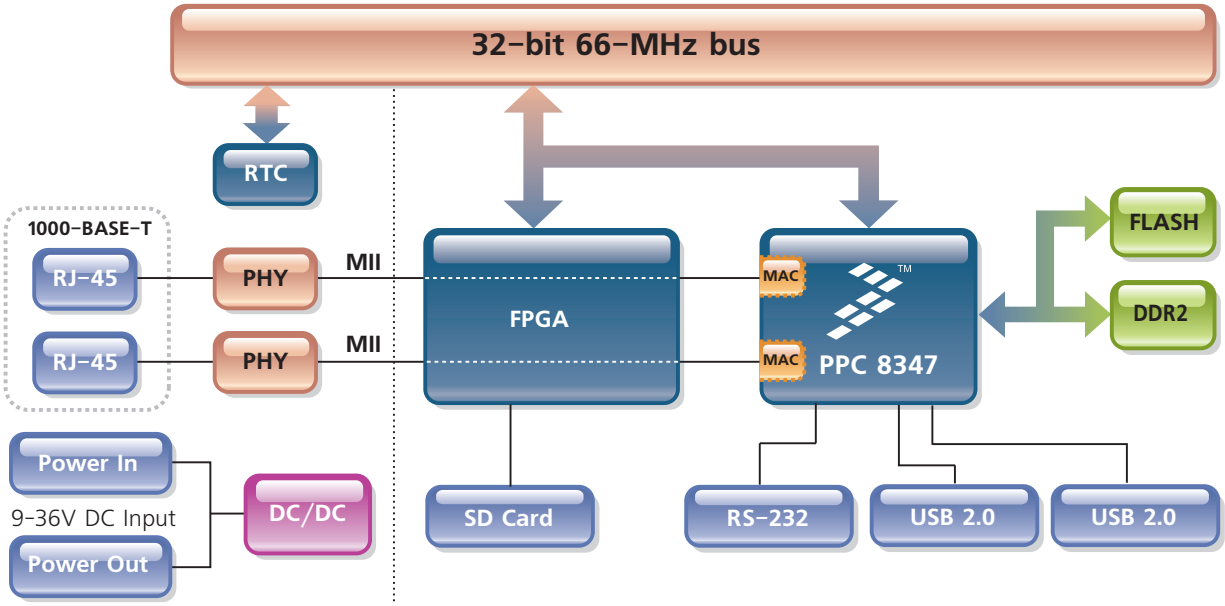

Figure 3-3. FreeScale PowerPC CPU/NIC Controller Architecture

The core of the system is a Freescale (formerly Motorola) PowerPC MPC8347 32-bit 400 MHz processor, which controls the following components:

- Primary Network Interface MII Port – NIC1
- Diagnostic Network Interface MII Port – NIC2
- RS-232 serial port
- UBS 2.0 dual port (Controller and Slave)
- 32MB flash memory
- 128MB of SDRAM
- SYNC port
- Control logic
- LEDs
- SD Card Slot (Card not included)
- Watchdog timer with real-time clock (battery backed)

Not all components are available for control from the CPU. The CPU can program flash memory, set the LEDs, set up the watchdog timer, set the real-time clock and use 256 bytes of backed-up memory in the watchdog timer chip. All functions are available at the firmware level only (described in iom.c/iom.h).

3.1.1 Primary Network Interface MII Port – NIC1

This port provides communication between the UEIPAC-300/600-1G system and the primary LAN network.

3.1.2 Secondary Network Interface Port – NIC2

This secondary port is available to the user for miscellaneous use during operation. This port may also be assigned as the primary Ethernet port if NIC1 is not available for use.

3.1.3 RS-232 Port

This port provides a serial communication link between the UEIPAC-300/600-1G system and a standard RS-232 terminal.

3.1.4 UBS 2.0 Dual Port (Controller and Slave)

The USB A and B ports are fully software supported.

3.1.5 32MB Flash Memory

The UEIPAC-300/600-1G system is provided with 32MB of flash memory.

3.1.6 128MB of SDRAM

The system is supplied with 128MB of SDRAM.

3.1.7 SYNC Port

A high-speed system-to-system synchronization connector permits triggers or clocks to be shared among multiple systems. Two systems may be connected together directly and larger groups may use the SYNC interface to share timing signals among many racks and systems.

3.1.8 SD Card

A slot for inserting a user-provided Secure Digital card is provided for on-board data storage. It can also store both data and Linux embedded programs using the UEI embedded toolkit. Supports FAT12, FAT16, and FAT32 file systems.

3.1.9 LEDs

The operating conditions indicated by the front panel LEDs are described in Figure 3-1 on page 21.

3.1.10 Watchdog Timer With Real-time Clock (Battery Backed)

The UEIPAC-300/600-1G system includes a watchdog timer with battery backed-up real-time clock.

Chapter 4 Real-time Operation with an IOM

This section discusses how to perform data mapping and streaming under control of a real-time operating system. The reason for making a separate chapter for real time operation is that writing real-time code can be done more efficiently without using the DQE. Therefore, this section discusses programming of streaming and data mapping operations at low-level.

4.1 Simple I/O

Simple I/O mode, which is commonly associated with lower speed systems, may also be used for real-time applications with a real-time operating system. The key requirement is not speed of operation but rather that all timing be deterministic and that no time deadline be missed.

4.2 Real-time Data Mapping (RtDmap)

Direct data mapping is a mechanism that allows you to create areas of input and output data that mirror data values on the input and output lines of networked IOMs. The following diagram illustrates the structure of DMap operation.

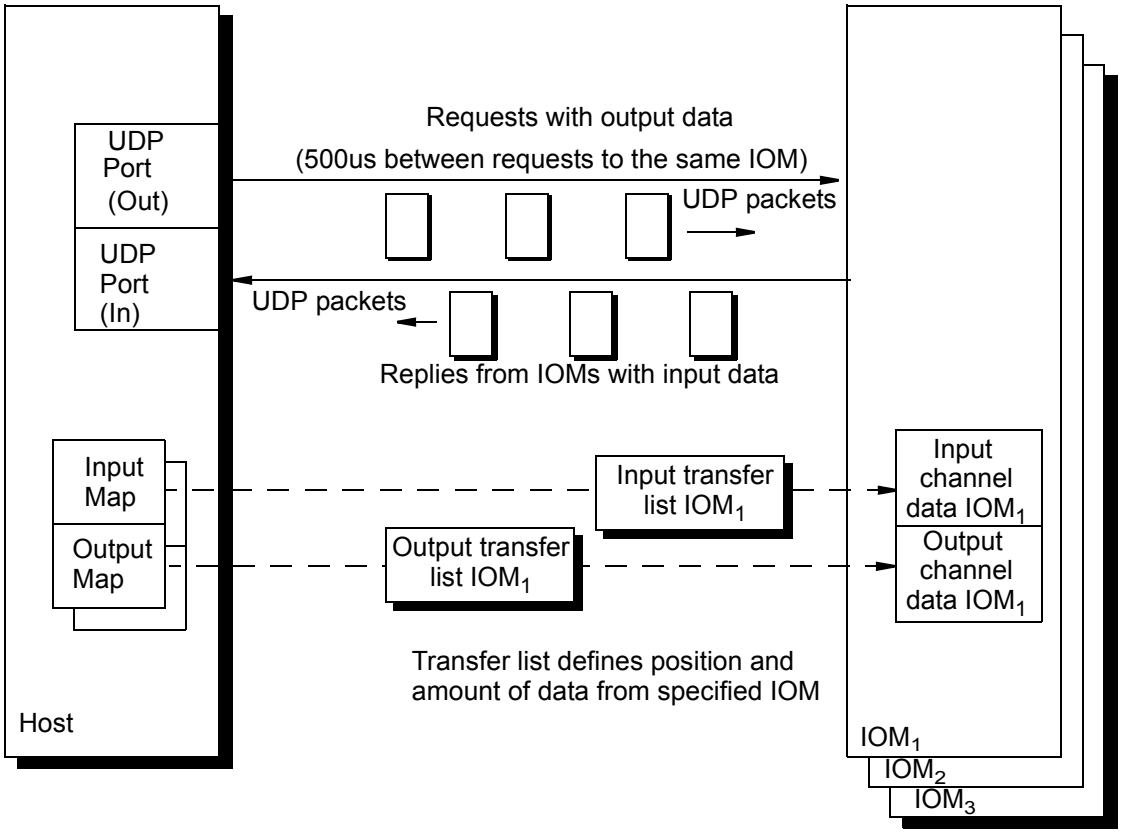

Figure 4-1. DMap Operation

Every DMap has its input and output maps and can work with a single multi-module IOM. Two DMaps can work with the same IOM, but must address different I/O boards (devices) within the IOM.

The maximum size of a DMap is limited to the size of a single packet – 510 bytes, which means that a DMap can be updated by receiving the data contained within a single new packet. Also, DMap allows representation of data either in raw or engineering units (volts by default).

In DMap mode, I/O devices perform at a rate sufficient to update input points fast enough to provide a fresh input reading with every reply packet. The output runs at a rate capable of updating outputs before the next portion of data arrives. Therefore, DMap mode meets the requirements of “hard” real-time operation.

4.2.1 Data Replication over the Network

DMap can be used for input data replication across a local area network if workstation NICs are set into promiscuous mode and receive all reply packets from the UDP interface. DMap can also be used in homogenous networks of IOMs in which IOMs exchange data between each other.

4.2.2 RtDmap Functional Description

The RtDmap API, described in this section, gives easy access to DMap operation without requiring use of the DQEngine. For more detailed information, refer to the PowerDNA Reference Manual.

Operation is as follows:

At each tick of the IOM clock, the IOM firmware scans the configured channels and stores the result in an area of memory called the DMap.

The host PC keeps its own copy of the DMap and synchronizes it periodically with the IOM’s version of the DMap. The rate at which the host transfers packets is controlled by the host and is usually set at a rate less than half the scan rate of the IOM clock.

This mode is very useful when the host computer runs a real-time operating system because it ensures that the host refreshes its DMap at deterministic intervals (hard real-time). It optimizes network transfer by packing all channels from multiple I/O boards into a single UDP packet, thus reducing the network overhead.

The standard (non-real-time) low-level API (**DqDmap***** functions) use the DqEngine (DQE) to refresh the DMap at a given rate and to retry a DMap refresh request if, for some reason, a packet is lost. Use of the DQE is necessary on desktop-oriented operating systems to ensure that the DMap is refreshed periodically, but is not required (and not recommended) for use with hard real-time operating systems.

The following is a list of the real time data mapping functions, with short descriptions of each. (Note that each of these functions does not use DQE.)

Table 4-1. RtDMap API Functions

Function	Description
`DqRtDmapInit`	Initializes the specified IOM to operate in DMAP mode at the specified refresh rate.
`DqRtDmapAddChannel`	Adds one or more channels to the DMAP.
`DqRtDmapGetInputMap`	Gets a pointer to the beginning of the input data map allocated for the specified device
`DqRtDmapGetInputMapSize`	Gets the size in bytes of the input map allocated for the specified device.

Table 4-1. RtDMap API Functions (Cont.)

Function	Description
DqRtDmapGetOutputMap	Gets a pointer to the beginning of the output data map allocated for the specified device.
DqRtDmapGetOutputMapSize	Gets the size in bytes of the output map allocated for the specified device.
DqRtDmapReadScaledData	Reads and scales the data stored in the input map for the specified device. **Note**: The data read is the data transferred by the last call to DqRtDmapRefresh(). This function should only be used with devices that acquire analog input data such as the AI-2xx series layers.
DqRtDmapReadRawData16	This function reads raw data from the specified device as 16-bit integers. **Note**: The data read is the data transferred by the last call to DqRtDmapRefresh(). This function should only be used with devices that acquire 16-bit wide digital data such as the AI-4xx series layers.
DqRtDmapReadRawData32	This function reads raw data from the specified device as 32-bit integers. **Note**: The data read is the data transferred by the last call to DqRtDmapRefresh(). This function should only be used with devices that acquire 32-bit wide digital data such as the DIO-4xx series layers.
DqRtDmapWriteScaledData	This function writes scaled data to the output map of the specified device. **Note**: The data written is actually transferred to the device on the next call to DqRtDmapRefresh(). This function should only be used with devices that generate analog data such as the AI-3xx series layers.
DqRtDmapWriteRawData16	This function writes 16-bit wide raw data to the specified device. **Note**: The data written is actually transferred to the device on the next call to DqRtDmapRefresh(). This function should only be used with devices that generate 16-bit wide digital data such as the DIO-4xx series layers.
DqRtDmapWriteRawData32	This function reads raw data from the specified device as 32-bit integers. **Note**: The data written is actually transferred to the device on the next call to DqRtDmapRefresh(). This function should only be used with devices that acquire 32-bit wide digital data such as the AI-4xx series layers.
DqRtDmapStart	This function starts operation and the IOM updates its internal representation of the map at the rate specified in DqRtDmapCreate.

Table 4-1. RtDMap API Functions (Cont.)

Function	Description
`DqRtDmapStop`	This function stops operation and the IOM stops updating its internal representation of the data map.
`DqRtDmapRefresh`	This function refreshes the host's version of the map by downloading the IOM's map. **Note**: The IOM automatically refreshes its version of the data map at the rate specified in `DqRtDMapInit()`. This function needs to be called periodically (a real-time OS is necessary) to synchronize the host and IOM data maps.
`DqRtDmapRefreshOutputs`	This function refreshes the host's version of the map by downloading the IOM's map. **Note**: The IOM automatically refreshes its version of the data map at the rate specified in `DqRtDMapInit()`. This function needs to be called periodically (a real-time OS is necessary) to synchronize the host and IOM data maps.
`DqRtDmapRefreshInputs`	This function refreshes the host's version of the map by downloading the IOM's map. **Note**: The IOM automatically refreshes its version of the data map at the rate specified in `DqRtDMapInit()`. This function needs to be called periodically (a real-time OS is necessary) to synchronize the host and IOM data maps.
`DqRtDmapClose`	This function frees all resources on the specified IOM allocated by the DMAP operation.

4.2.3 RtDmap Typical Program Structure

The following is a quick tutorial on use of the RtDmap API (with error handling omitted):

1. Initialize the DMAP to refresh at 1000 Hz.
```
DqRtDmapInit(handle, &dmapid, 1000.0);
```
2. Add channel 0 from the first input subsystem of device 1.
```
chentry = 0;
DqRtDmapAddChannel(handle, dmapid, 1, DQ_SS0IN,
&chentry, 1);
```
3. Add channel 1 from the first output subsystem of device 3.
```
chentry = 1;
DqRtDmapAddChannel(handle, dmapid, 3, DQ_SS0OUT,
&chentry, 1);
```
4. Start all devices that have channels configured in the DMAP.
```
DqRtDmapStart(handle);
```
5. Update the value(s) to be output to device 3.
```
outdata[0] = 5.0;
DqRtDmapWriteScaledData(handle, dmapid, 3, outdata, 1);
```
6. Synchronize the DMAP with all devices.
```
DqRtDmapRefresh(handle, dmapid);
```

© Copyright 2009
United Electronic Industries, Inc.

7. Retrieve the data acquired by device 1.
   ```
   DqRtDmapReadScaledData(handle, dmapid, 1, indata, 1);
   ```
8. Stop the devices and free all resources.
   ```
   DqRtDmapStop(handle, dmapid);
   DqRtDmapClose(handle, dmapid);
   ```

4.3 Real-time Variable-size Data Mapping (RtVmap)

This feature is similar to RealTime DMap operation (see “Real-time Data Mapping (RtDmap)” on page 24) except that the size of the data transfer is variable.

The RtVmap API, like the RtDmap API, gives easy access to the VMap operating mode without needing the DqEngine.

VMap is a protocol developed for *control* applications in which the ability to get immediate real-time data may be more important than receiving a continuous gapless flow of the data. VMap is also well-suited for many real-time messaging applications, as described below.

Messaging layers are normally supported by the Msg protocol, which shares the same buffering mechanism as the ACB protocol. The Msg protocol buffer receives packets and delays releasing newer packets to the user application until it re-requests and receives all the packets in the previous message stream. Although this protocol does provide a gapless stream of messages, it is not suited for real-time operation because timing is not determnistic.

VMap, however, can provide a real-time alternative to the Msg protocol for messaging devices — at the expense of restricting the ability to recover lost packets. It shifts the decision about whether or not to recover the lost packet to the user application. A set of hard real-time VMap functions is listed below in **Table 4-2**.

At high level, VMap is very similar to DMap. A user creates a VMap with output and input buffers and add channels/layers of interest to it. VMap packets also have additional fields. First of all, there is a flag field required to guarantee continuity of messaging data. Second, an output buffer adds a pair of fields for each channel in the map at its header. The first field provides the IOM with information on how much data is to be transmitted for that channel; the second field defines the maximum size of data to be received from that channel. The offsets of the output data in the buffer should be in agreement with the size of the data in the buffer header.

An input packet also contains a flag field as well as the number of bytes actually written, actually received plus (optionally) the number of bytes available in the receive FIFO, and the room available in the transmit FIFO. This feature allows flexibility in allocating packet slices for various channels. Each time packets are exchanged between host and IOM, the user application can select different sizes for outgoing and incoming data, taking into consideration the amount of data required to be sent and the size of data accumulated in the receiving FIFO. If you don’t use a channel at this time, you should set *size to send* and *size to receive* to zero. The header has a fixed width set up before starting VMap operation; the header size cannot be changed on the fly even if the channel is no longer in use.

Note that VMap has a function that returns the VMap ID to the user for use in systems that have multiple IOMs. Since packets from multiple IOMs may be received by the host out of time sequence, this function gives the host the information necessary to call the right VMap processing routine for that packet.

© Copyright 2009
United Electronic Industries, Inc.

Table 4-2 is a list of the real-time variable data mapping functions, with short descriptions of each. Refer to the PowerDNA Reference Manual API for more detailed information.

Table 4-2. RtVmap API Functions

Function	Description
`DqRtVmapInit`	Initializes the specified IOM to operate in VMap mode at the specified refresh rate.
`DqRtVmapAddChannel`	This function adds a channel to `<vmapid>` VMap. The function adds an entry to the transfer list. Channels with an SSx_IN subsystem are added to the transfer list; channels with an SSx_OUT subsystem are added to the output transfer list. Channel in `<cl>` should be defined in the standard way including channel number, gain, differential, and timestamp flags. Configuration `<flags>` for the input subsystem can include `DQ_VMAP_FIFO_STATUS` to report back the number of samples in the input FIFO waiting to be requested (after output packets are processed). Configuration `<flags>` for the output system can include `DQ_VMAP_FIFO_STATUS` to report back the number of samples that can still be written into the output FIFO before it becomes full (after all transmitted bytes have been written). Note that this flag adds a uint16 word to the standard header for an input packet, thus inceasing te size of the header and decreasing the size available for data. `<clSize>` specifies the maximum number of array entries. The Output VMap buffer, which transfers data from host to IOM, has the structure shown in Table 4-3 on page 32.

Table 4-2. RtVmap API Functions (Cont.)

Function	Description
DqRtVmapAddChannel (cont.)	The total length of the buffer cannot exceed the size available in the UDP packet minus the combined size of the DQPKT and DQQRRD headers. The output buffer of VMap contains information to be written to the channel output FIFOs of the messaging layer (as well as theanalog or digital layers equipped with hardware FIFOs). It also specifies the number of bytes to read from the same channel, if any. Data for or from the channel should be assembled in accordance with the message structure of that layer. Flags are used to make data ready and to acknowledge packet execution. This feature arises because VMap relies on continuous data flow compatible with messaging layers as well as continuous acquisition and output and thus must ensure continuuty of data. In other words, no message can be sent or received twice. The Input VMap buffer, which transfers data from IOM to host, has the structure shown in Table 4-4 on page 33. The Input VMap buffer contains information showing how much data was actually retrievded from the channel FIFO and how much of the data in the output buffer has been written to that channel. The header size cannot be changed after `DqRtVmapStart()` is called. In other words, after a channel is added using `DqRtVmapAddChannel()`, the header size increases by one in the output packet and by one or two (if `DQ_VMAP_FIFO_STATUS` is set) uint16 words in the input packet. The header allocation cannot be changed until the current VMap is destroyed and a new one is created. If youwould like to send zero bytes for that channel or receive zero byttes froma a channel, VMap fills the appropriate header field with 0. **Note:** Each call to `DqRtVmapAddChannel()` adds one or more transfer list entries. Ther indices are zero-origin, sequential, and cumulative. For example, if one adds five channels in the first call to this function, the transfer list index of the last channel is 4. For the next call, the last channel will have transfer list index equal to 9.
DqRtVmapStart	This function sets up all parameters needed for operation – channel list and clock; transfers and finalizes the transfer list. The function also parses the transfer list and stores offsets of the headers for each transfer list entry. If clocked devices (AIn/AOut) are used, the function programs devices at the rate specified in `DqRtDmapInit`.
DqRtVmapStop	This function stops operation and the IOM stops updating its internal representation of the data map.

Table 4-2. RtVmap API Functions (Cont.)

Function	Description
DqRtVmapClose	This function destroys the <vmapid> VMap.
DqRtVmapRefresh	This function refreshes the host version of the map by downloading the IOM map. Use the DQ_VMAP_REREQUEST flag if you want to re-request the failed transaction instead of performing a new one. In such case, the dqCounter in the DQPKT header will not be incremented by the host and the IOM will not output/input a new message if the IOM already processed it (reply packet lost). Instead, the IOM will reply with a copy of the previous packet. If the IOM never received the packet, it will process it in the normal way. **Note:** The IOM automatically refreshes its version of the data map at the rate specified in DqRtVMapInit(). This function should be called periodically (a real time OS is required) to synchronize the host and IOM data maps).
DqRtVmapRefreshOutputs	This function refreshes the host version of the map by downloading the IOM map. Use DQ_VMAP_REREQUEST flag if you want to re-request the failed transaction instead of performing a new one. **Note**: This function needs to be called periodically (real-time OS is required) to synchronize host and IOM data.
DqRtVmapRefreshInputs	This function refreshes the host version of the map by downloading the IOM map. **Note**: This function needs to be called periodically (a real-time OS is necessary) to synchronize the host and IOM data maps.
DqRtVmapGetInputPtr	This function gets the pointer to the beginning of the input data allocated for the specified entry. **Note**: This function can be called only after packet is received.
DqRtVmapGetOutputPtr	This function gets the pointer to the beginning of the output data allocated for the specified entry. **Note:** This function can be called only after transmission size for all channels is written.
DqRtVmapGetInputMap	Get pointer to the beginning of the input data map allocated for the specified device. **Note:** This fuunction can be called only after a packet is received, because the actual positions of the input data in the packet for each transfer list entry depend on the number of bytes actually retrieved from the input FIFO. If the number of bytes retrieved is less than requested, VMap will not waste the space in the packet, but rather will pack it to decrease transmission time.

Table 4-2. RtVmap API Functions (Cont.)

Function	Description
`DqRtVmapGetOutputMap`	This function gets the pointer to the beginning of the output data map allocated for the specified entry. **Note:** This function can be called only after transmission size for all channels is written. Actual offsets of the data for each channel in the output packet depend on the size of the data stored in the packet header. Thus, this function makes sense only if all data is placed into the packet.
`DqRtVmapAddOutputData`	This function copies data into the output packet and returns the number of bytes left in the packet. **Note:** This function modifies the output packet.This function must be called *before* `DqRtVmapRefresh()`.
`DqRtVmapRqInputDataSz`	This function requests the number of bytes to receive in the input packet. It returns the number of bytes left in the buffer, the actual size requested, and the pointer to the location where the data will be stored. **Note:** This function modifies the output packet.This function must be called *before* `DqRtVmapRefresh()`.
`DqRtVmapGetInputData`	This function copies data from the input packet and returns the number of bytes copied and the size available in the input FIFO. **Note:** This function must be called *after* `DqRtVmapRefresh()`.
`DqRtVmapGetOutputDataSz`	This function examines the input packet and returns the number of bytes copied from the output packet to the output FIFO and (optionally) how much room is available in the output FIFO. **Note:** This function must be called *after* `DqRtVmapRefresh()`.

Table 4-3. Output VMap Buffer

Size	Flags (uint16)
Size to write to Ch0 (uint16)	Size to write to ChN (uint16)
•	•
•	•
•	•.
Size to read from Ch0 (uint16)	Size to read from ChN (uint16)
Data for Ch0 (of specified size)	
•	
•	
•	
Data for ChN (of specified size)	

Table 4-4. Input VMap Buffer

Size	Flags (uint16)
No. of bytes retrieved from Ch0 (uint16)	No. of bytes remaining in Ch0 (uint16, optional)
• • •	• • •
No. of bytes retrieved from ChN (uint16)	No. of bytes remaining in ChN (uint16, optional)
No. of bytes written to Ch0 (uint16)	No. of bytes that can be written to Ch0 (uint16, optional)
• • •	
No. of bytes written to ChN (uint16 optional)	No. of bytes that can be written to ChN (uint16, optional)
Data from Ch0 (of specified retrieved size)	
• • •	
Data from ChN (of specified retrieved size)	

4.3.1 RtVmap Typical Program Structure

The following is a short tutorial example that uses the RtVmap API (handling of error codes is omitted):

1. Initialize the VMAP to refresh at 1000 Hz:
```
DqRtVmapInit(handle,&vmapid,1000.0);
```
2. Configure device input output ports using the appropriate DqAdv*** function. For example, the following configures an ARINC-429 device (DEVN) input and output ports 0 to run at 100kbps with no parity and no SDI filtering.
```
DqAdv566SetMode(handle, DEVN, DQ_SS0OUT, 0,
DQ_AR_RATEHIGH | DQ_PARITY_OFF);
DqAdv566SetMode(handle, DEVN, DQ_SS0IN, 0,
DQ_AR_RATEHIGH|DQ_PARITY_OFF|DQ_AR_SDI_DISABLED);
```
3. Add input port 0 to VMAP, set flag to retrieve the status of the input FIFO after each transfer:
```
chentry = 0;
flag = DQ_VMAP_FIFO_STATUS;
DqRtVmapAddChannel(handle, vmapid, DEVN, DQ_SS0IN,
&chentry, &flag, 1);
```
4. Add output port 0 to VMAP, set flag to retrieve the status of the output FIFO after each transfer.
```
chentry = 0;
```

```
flag = DQ_VMAP_FIFO_STATUS;
DqRtDmapAddChannel(handle, vmapid, DEVN, DQ_SS0OUT,
&chentry, &flag, 1);
```

5. Enable ARINC-429 ports.
   ```
   DqAdv566Enable(handle, DEVN, TRUE);
   ```
6. Start all devices that have channels configured in the VMAP.
   ```
   DqRtVmapStart(handle, vmapid);
   ```
7. Prepare ARINC word to send through port 0 and update VMAP.
   ```
   uint32 arincWord = DqAdv566BuildPacket(data, label,
   ssm, sdi, parity);
   DqRtVmapAddOutputData(handle, vmapid, 0,
   sizeof(uint32), &accepted, (uint8*)&arincWord);
   ```
8. Specify that we wish to receive up to MAX_WORDS words received by port 0.
   ```
   DqRtVmapRqInputDataSz(handle, vmapid, 0,
   MAX_WORDS*sizeof(uint32), &rx_act_size, NULL);
   ```
9. Synchronize the VMAP with all devices.
   ```
   DqRtVmapRefresh(handle, vmapid, 0);
   ```
10. Retrieve the data received by port 0.
    ```
    uint32 recvWords[MAX_WORDS];
    DqRtVmapGetInputData(handle, vmapid, 0,
    MAX_WORDS*sizeof(uint32), &rx_data_size, &rx_avl_size,
    (uint8*)recvWords);
    ```
11. We can also check how much data was actually transmitted during the last refresh.
    ```
    DqRtVmapGetOutputDataSz(handle, vmapid, 0,
    &tx_data_size, &tx_avl_size);
    ```
12. Stop the devices and free all resources.
    ```
    DqRtVmapStop(handle, vmapid);
    DqRtVmapClose(handle, vmapid);
    ```

Appendix A

Field Replacement of Fuses on DNA Boards Used in UEIPAC Cubes

Some boards used in UEI DAQ I/O systems require field replacement of fuses when unexpected overloads occur. Locations of these fuses are shown in **Figure A-1** through **Figure A-2**. Part numbers for the replacement fuses are listed **Table A-1**.

Table A-1. DNA/DNR Replacement Fuses

UEI Fuse ID (Board)	**Rating**	**UEI Part No.**	**Description**	**Mfr.**	**Mfr P/N**
F1	5A	925-5125	FUSE 5A 125V SLO SMD SILVER T/R	Littlefuse	0454005.MR
F2	5A	925-5125	FUSE 5A 125V SLO SMD SILVER T/R	Littlefuse	0454005.MR
F3 (1GB)	10A	925-1125	FUSE 10A 125V FAST NAN02 SMD	Littlefuse	0451010.MRL

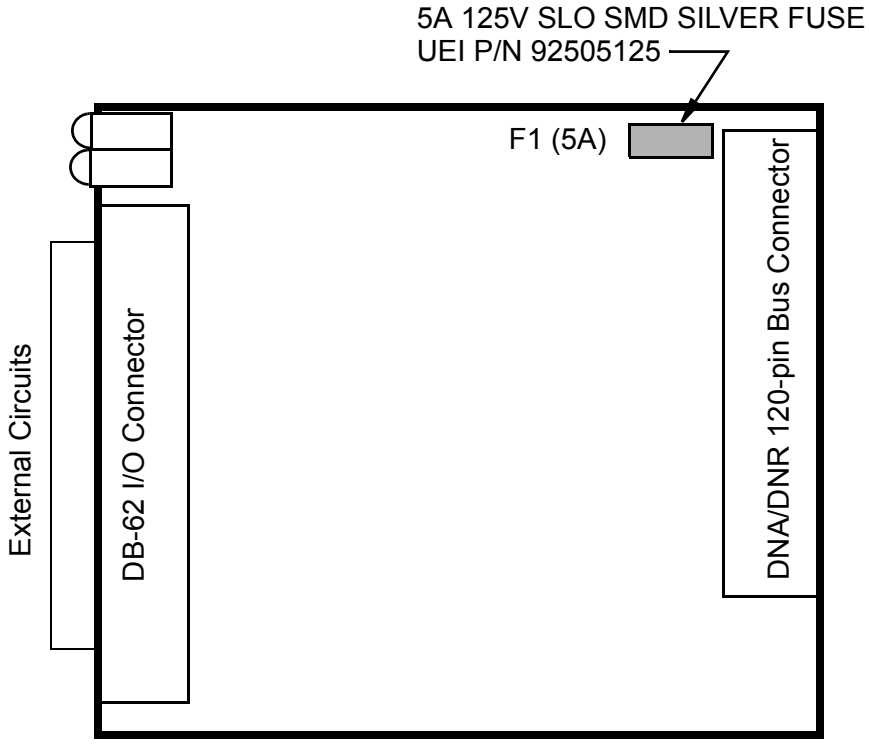

Figure A-1. Location of Fuse for PL-61x, PL-62x, and PL-63x Boards

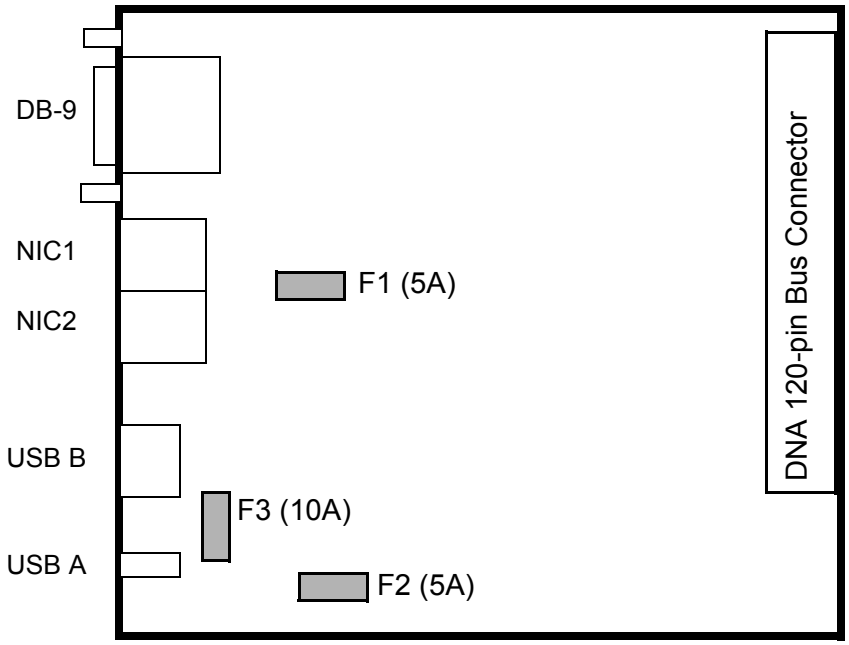

Figure A-2. Location of Fuses for DNR-POWER-1GB Board

Index

C
Conventions 2

D
DNR Core Module
Device Architecture 22
DNR-CPU-1000 Core Module 21

F
Field Connections 19
Fuse Replacement 1

M
Mounting 19

O
Organization of Manual 1

R
Real-time Operation 24

S
Specifications 5
Support ii
Support email
support@ueidaq.com ii
Support FTP Site
ftp //ftp.ueidaq.com ii
Support Web Site
www.ueidaq.com ii

© Copyright 2008 all rights reserved
United Electronic Industries, Inc.